# DataONE RunManager: a scientific workflow provenance data management tool


## ABSTRACT

In this demo, we demonstrate a scientific workflow provenance data management tool – DataONE RunManager that can capture, store, query, visualize, and publish of a single Matlab script or multiple Matlab script runs. The important features of DataONE RunManager include: (1) There are three forms of data provenance supported by RunManager: prospective provenance, retrospective provenance, and hybrid provenance; (2) Query-based approach is used for data dependency analysis. For a workflow project, we have multiple provenance graphs consisting of a graph of prospective provenance, a graph of hybrid provenance, a graph of retrospective multiple_scripts_multiple_runs provenance.



## KEYWORDS

prospective provenance, hybrid provenance, retrospective provenance, provenance query

**ACM Reference format:**
. 2016. DataONE RunManager: a scientific workflow provenance data management tool. In *Proceedings of ACM Conference, Chicago, USA, June 2017 (SSDBM'17),* 6 pages.
DOI:


## 1 INTRODUCTION

Provenance from scripts and runs of scripts plays an important role in modelling, designing, debugging, testing, reliability and sharing. A considerable amount of research has been done on investigating methods of harvesting provenance information from scripts and runs of scripts, ranging from conventional approaches, e.g. research compendium (folder layouts) and logging to recent provenance tools, such as YesWorkflow [7], noWorkflow [8], RDataTracker [6], Reprozip [3], RunManager [2]. This demo will illustrate the prospective and retrospective provenance that are captured by YesWorkflow and DataONE RunManagers, and how to query, and visualize using a combination of tools for exposing both prospective, retrospective and hybrid provenance.

The DataONE RunManager is a scientific workflow provenance data management tool. It can capture, store, query, visualize, and publish of one or multiple Matlab or R script runs. YesWorkflow (YW) is a prospective provenance tool allowing users to model and design prospective provenance via YW tags. There are three types of provenance demonstrated in our demo: prospective provenance, retrospective provenance, and hybrid provenance. The retrospective provenance captured during a script execution includes information about the script that was run, files that were read or written, and details about the execution environment at the time of execution.


*SSDBM'17, Chicago, USA*



DOI:


DataONE RunManager provides two implementations: the `recordr` R package and the `matlab-dataone` toolbox. The RunManagers can capture retrospective provenance information when a running of R or Matlab script and generate a data package. A DataONE package includes scripts, a list of input files, and a list of generated files, science metadata that are associated with the run. In addition, a DataONE data package can be efficiently archived so that the past versions of files can be retrieved for a run in order to investigate previous versions of processing or analysis, support reproducibility, and provide an easy way to publish data products and all files that contributed to those products to a data repository such as the DataONE network.

Then, we show three forms of data provenance. Particularly, we present multi_scripts_multi_runs provenance in DataONE provenance database. From the multi_scripts_multi_runs provenance, it enables a longitudinal view of a typical real-life scientific workflow that consists of multiple phases. Since computational and data science experiments can often last days, weeks, or even months and often require the execution of multiple scripts or workflows with varying input datasets and parameters, some of these script runs appear as chained together implicitly via intermediate data. Multiple_scripts_multi_runs provenance graphs can be joined on file path, file content, etc.

We use query-based approach for provenance analysis. A query is implemented in Prolog and SQL. A graph is used to represent the query results. For a workflow project, we have multiple provenance graphs consisting of a graph of prospective provenance, a graph of hybrid provenance, a graph of retrospective multi-run provenance.

The rest of this paper is organized as follows. The three forms of data provenance used in RunManagers are presented in Section 2. Section 3 discusses two example use cases whereas Section 4 concludes the paper.

**Related Work**. Table 1 presents a representative set of related provenance tools for provenance collection. YesWorkflow provides prospective graph revealing the data dependencies at the conceptual workflow level. URI-template specifies a file path pattern while user-defined log-files capture runtime provenance observables at any level of granularity. The noWorkflow system (NW)[8] captures code-level retrospective provenance by employing a Python profiling library to obtain Python function call chains and variable assignments. The DataONE RunManagers[2] overload and thus intercept file I/O operations to automatically capture runtime observables at the file level.

## 2 THREE FORMATS OF PROVENANCE

`Data provenance` is metadata that describes the origin and processing history of a data artifact. In the computational and data sciences, we can distinguish three forms of data provenance, i.e., `prospective provenance`, `retrospective provenance`, and `hybrid provenance`. The prospective provenance describes the general

**Table 1: Various Provenance Tools and Methods Breakdown**

| | Language: Application / Implementation | Scope | Methods |
|---|---|---|---|
| YesWorkflow | Any / Java | Prospective Provenance | Annotations added by users; language independent |
| URI-templates | Any / Any | Retrospective Provenance | Templates matching with runtime file locations; language independent |
| User-defined log-files | Any / Any | User-defined Retrospective Provenance | Exported runtime variables and files provenance by users |
| noWorkflow | Python / Python | Retrospective Provenance | Runtime function calls, variables, file inputs/outputs, execution environmental information |
| RunManagers | R / R<br>MATLAB / MATLAB | Retrospective Provenance | Runtime file inputs/outputs, execution environmental information |

workflow by which data is produced, while the retrospective provenance consists of runtime observables, e.g., read and generated files during a script run (coarse-grained observables). Fine-grained provenance can be captured as well, e.g., changes to individual data records, program variables, parameter settings, etc. Hybrid provenance can be defined as a graph containing an alternation of process steps, situated in the context of the prospective workflow graph, and of the specific instances of data elements that have been generated and consumed by executions of those steps. The data elements can be instantiated to document the specific files used in a concrete workflow run.

**Listing 1: YW annotation block in C3C4 Matlab script**

```
%% Load input: long-term monthly mean precipitation data
% @BEGIN fetch_monthly_mean_precipitation_data
% @in mean_precip @URI
% file:inputs/narr_apcp_rescaled_monthly/
% apcp_monthly_{start_year}_{end_year}_mean.{month}.nc
% @out Rain @AS Rain_Matrix
Rain=zeros(ncols,nrows,12);
for m=1:12
   rncid=netcdf.open(strcat(
    'inputs/narr_apcp_rescaled_monthly/monthly_2000_2010_mean.',
    num2str(m),'.nc'), 'NC_NOWRITE');
   rvid=netcdf.inqVarID(rncid, 'apcp_monthly_mean');
   Rain(:,:,m)=netcdf.getVar(rncid,rvid);
   netcdf.close(rncid)
end
% @END fetch_monthly_mean_precipitation_data
```

YesWorkflow (YW) is a prospective provenance tool allowing users to easily recover high-level workflow models latent in scripts using simple user-annotations embedded as script comments[7]. List 1 shows a code snippet from a MATLAB script with YW annotations. The @begin and @end tags are used to mark code blocks (i.e., processing steps in the YW model); @in and @out tags are used to define the dataflow between blocks; the @uri template (which is paired with an @in tag) is used to define a where the data file is located. Note that @uri templates define a metadata pattern via so-called template variables (here: *start_year*, *end_year*, and *month*). In this way, the script author can document the folder structure and file-naming conventions used by the script. List 2 shows a YW annotation block modelling a dataflow for `lsp_status_file` in a R script. The optional @param tag is used to specify the input and output parameters.

**Listing 2: YW annotation block in one OHIBC R script**

```
# @BEGIN output_estimate_status_file_by_region_since_1980
# @param year_span
# @param dir_goal
# @in area_protected_total_file @URI
      file:{dir_goal}/output/area_protected_total.csv
# @in status_file_handle
# @out status_df_data
status_df <- read.csv(file.path(dir_goal, 'output',
      'area_protected_total.csv')) %>%
  select(rgn_id, year, lsp_status) %>%
  filter(year > max(year) - year_span)
# @END output_estimate_status_file_by_region_since_1980

# @BEGIN write_status_to_csv_file
# @in status_df_data
# @out lsp_status_file @URI file:{dir_goal}/output/lsp_status.csv
write.csv(status_df, status_file)
# @END write_status_to_csv_file
```

A scientific workflow script can run inside of RunManager by invoking the `record` function. The RunManager can automatically capture file-level retrospective provenance by overloading and thus intercepting file I/O operations. The automated approach is efficient and useful for scientists. RunManagers support the following APIs: `record`, `startRecord`, `endRecord`, `listRuns`, `deleteRuns`, `viewRun(s)`, `traceRuns` for R, `plotRuns` for R and `publishRun` in order to capture, search, archive, visualize, and publish a script run. Then, the captured provenance information is saved to local database. The database tables include: `execution metadata`, `file metadata`, `tag`, `module dependencies` for MATLAB and `provenance relationships` for R. In addition, the provenance database can be exported as Prolog facts. Retrospective provenance graphs can be rendered from the exported Prolog facts. With YesWorkflow and RunManager, scientists can model, design, execute, debug, re-configure and re-run their analysis and visualization pipelines. Provenance is useful for a scientist to interpret their workflow results and for other scientists to establish trust in the experimental result [1].

For example, a multi_scripts_multi_runs provenance sub-graph can be produced by executing a Prolog query [1] which consists of four Prolog queries using the exported Prolog facts. A example set of logical rules are highlighted in List 3. The joining operation is based on file content (hash code). For the recursive rules

[1] https://github.com/yesworkflow-org/yw-idcc-17/blob/master/OHIBC_Howe_Sound_project/multi_runs_retrospective_provenance_queries/queries/render_rm_graph_upstream_of_file_q1_sha256.sh

rm_file_downstream computes all upstream files for the given file Sha256_2. It has two base cases: one is the file object is itself downstream file object and the other is immediate derived relationship.

**Listing 3: Prolog rules for multiple scripts multiple runs provenance**

```
:- table rm_file_immediately_downstream/2.
rm_file_immediately_downstream(Sha256_1, Sha256_2) :-
    exec_used_files(_, E, _, _, _, Sha256_1, _, _, _),
    exec_generated_files(_, E, _, _, _, Sha256_2, _, _, _).

# Base case 1.
:- table rm_file_downstream/2.
rm_file_downstream(Sha256_1, Sha256_2) :-
    Sha256_1 = Sha256_2.
# Base case 2.
rm_file_downstream(Sha256_1, Sha256_2) :-
    rm_file_immediately_downstream(Sha256_1, Sha256_2).
# Recursive part.
rm_file_downstream(Sha256_1, Sha256_2):-
    rm_file_immediately_downstream(Sha256_1, S),
    rm_file_downstream(S, Sha256_2).

# The rule gv_nodes__files__upstream_of_file_sha256 computes all
      file objects which has a given hashcode (DataSha256).
gv_nodes__files__upstream_of_file_sha256(_, DataSha256) :-
    rm_file_downstream(Sha256, DataSha256),
    filemeta(_,_,FilePath,Sha256,_,_,_,_,_,_,_),
    gv_labeled_node_sha256(Sha256, FilePath),
    fail
    ;
    true.
# The rule gv_edges__execution_to_file__upstream_of_file_sha256
      draws edges from an execution object to file objects by first
      finding all upstream file objects for a given hashcode
      (DownstreamDataSha256).
gv_edges__execution_to_file__upstream_of_file_sha256(_,
      DownstreamDataSha256) :-
    rm_file_downstream(DataSha256, DownstreamDataSha256),
    exec_generated_files(_,ExecutionId,_,_,_,DataSha256,_,_,_),
    gv_unlabeled_edge_sha256(ExecutionId, DataSha256),
    fail
    ;
    true.
```

Hybrid provenance can be derived using logic rules. The approach of creating hybrid provenance is as follows: after a script run is completed, the retrospective provenance information are collected and populated to Sqlite provenance database. Then, Matlab RunManager has a function to export content of tables filemeta as a list of input and output file paths that are fed to yw-matlab bridge. The bridge will match the URI templates with the exported file list and the logical rule for producing hybrid provenance is shown in List 4.

**Listing 4: Prolog rules for computing hybrid provenance**

```
:- table data_uri_resource/2.
data_uri_resource(DataId, ResourceURI) :-
    data_resource(DataId,ResourceId),
    resource(ResourceId, ResourceURI).
```

After obtaining three forms of data provenance, we can demonstrate the provenance dependency with queries. We have developed both SQL queries and Prolog rule-based queries (e.g., List 3 and 4) against three forms of provenance. For example: What outputs does the workflow (script) have? Advanced provenance queries can take the form of regular path queries (RPQs), tree-structure queries, or graph queries and show the data lineage hidden in the provenance information. For example: (1) What are the ancestors or descendants of a given output in a prospective provenance graph? (2) What are the ancestors or descendants of a given output in a retrospective- or hybrid- provenance graph? (3) Which possible execution (i.e., dataflow) paths satisfy a given regular path expression? The queries and results are demonstrated with two use cases in Section 3.

## 3 EXAMPLE USE CASES

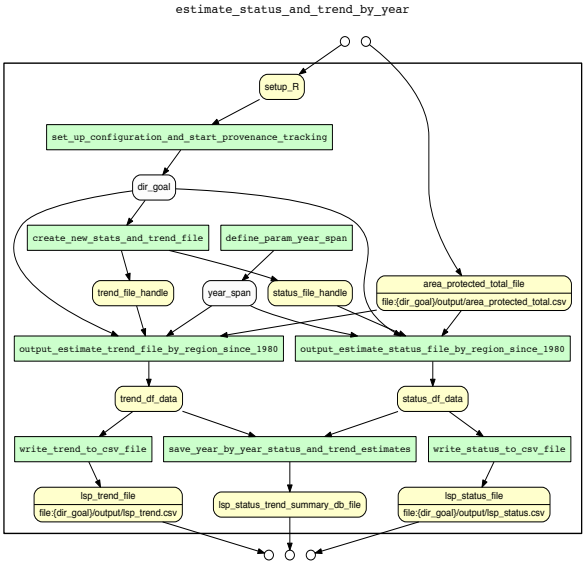
In this demo, two real-life scientific examples are used: (A) a workflow for Ocean Health Index (OHI) for Howe Sound, British Columbia[4]; (B) a workflow for Carbon3/Carbon4 (C3/C4) soil mapping for North America[5, 9]. The OHIBC example calculates the region's status based upon percent of protected area within 1 km inland buffer and percent of protected area within 3 nautical mile offshore buffer, compared to a reference point of 30% protected area. The data sources used in the OHIBC workflow include: (1) BC-specific WDPA dataset that was created from WDPA global dataset, then rasterize to BC Albers at 500m resolution; (2) BC Parks, Ecological Reserves, and Protected Areas (PEP) data. The OHIBC workflow creates two outputs: one file is for estimate of status by region since 1980 (lsp_status.csv) and the other file is for estimate of trend by region since 1980 (lsp_trend.csv). The OHIBC Howe Sound workflow runs in a batch mode.

RunManager can show the prospective provenance and retrospective provenance as graphs for one script. There are eleven R scripts in the OHIBC example. A user can add YW tags to these R scripts to show prospective provenance graphs. Then, the user can start to run the OHIBC example using RunManager. After the run was done, the retrospective provenances (e.g., the execution of each script, used files, generated files including file path, file content, size, and archived file path) are captured and stored in the database. After exporting retrospective provenance as a list of input and output file paths, hybrid provenance can be derived by joining retrospective provenance and prospective provenance facts using shared resource ids. Figure 1 depicts the prospective and hybrid graphs.

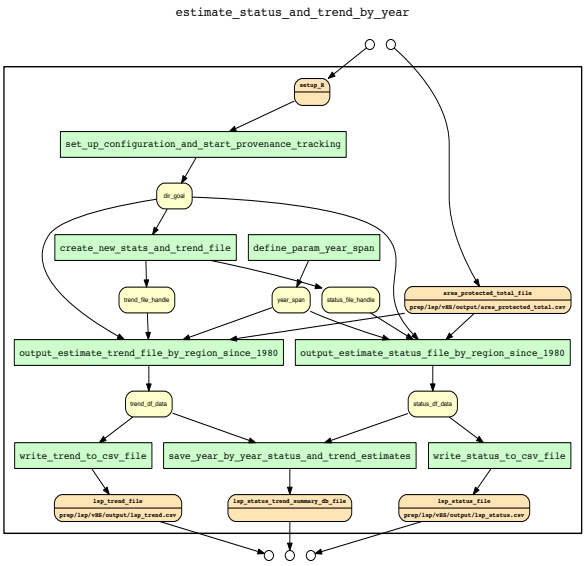
RunManager can review the overview graph for all script runs because RunManager have recorded the execution history for the OHIBC example. Hence, using R RunManager provenance database, the plotRun() API gives us an overview for the OHIBC Howe Sound workflow example shown in Figure 2. Alternatively, using exported Prolog facts from the provenance database, we can use prolog queries to render a complete multi_scripts_multi_runs graph. With exported Prolog facts, we can use query_based approach to do provenance dependency analysis which will be explained in the next section.

**Provenance Query Examples**. The first category provenance query is prospective provenance queries can tell us if an output depends a particular input or if an output depends on all of inputs, see the script estimate_status_and_trend_by_year.R in Figure 1).

The second category provenance query is hybrid provenance query. In a hybrid provenance graph, the "workflow backbone" is given by the user-declared YW model (prospective provenance) and the execution details are filled in from one or more sources of




**(a) prospective provenance graph**




**(b) hybrid provenance graph**

**Figure 1: Complete YW prospective and hybrid provenance graphs for the last step script `estimate_status_and_trend_by_year.R`. The green box represents a computation step; a round yellow square box represents a data element declared in the script via YW tags; an orange box represents a data element which URI is expanded with the runtime file path.**

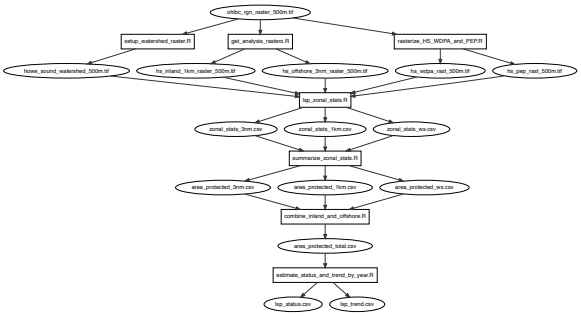



**Figure 2: Complete retrospective provenance graph for the OHI Howe Sound workflow example plotted by the API `plotRun()` in DataONE R RunManager**

runtime observables (retrospective provenance). Figure 1 shows a hybrid provenance graph for the last script in the OHIBC example `estimate_status_and_trend_by_year.R`. We can see that the yellow round box changes to orange round box and the file path pattern embedded is expanded with runtime relative file paths. Please visit the web link for prospective and hybrid provenance queries [2].

The third category provenance query is `multi_scripts_multi_runs provenance query`. RunManager retrospective provenance queries refer to the runtime retrospective provenance obtained from executing multiple scripts, using RunManager. Since the scripts in a project are usually logically connected through sharing some of their inputs and outputs, the corresponding provenance traces are also connected through shared files. Figure 2 shows one complete graph for the OHIBC workflow example using the R provenance relational database and Figure 3 shows a upstream retrospective provenance subgraph that is relevant to "`lsp_trend.csv`" output using exported facts. The `multi_scripts_multi_runs` query can be found at [3].

The second example script (in MATLAB) produces Carbon 3/Carbon 4 (C3/C4) soil maps for North America using average rain and air temperature monthly data from year 2000 to 2010. There are two versions for C3C4 example. In the version 1, there is one C3C4 example. The captured provenance and query results for C3C4 version is available at [4]. In order to simulate real-life workflow, we create a version 2 C3/C4 example by splitting one script to three phases: `preparing_step`, `examining_pixels_for_grass` and `generating_final_results`. After `preparing_step`, a user can run the second script `examine_pixels_for_grass` for multiple times with different parameters, then writes to the output files in `generating_final_results`. We present such retrospective provenance graphs based on file content (hash code) in Figure 4 which records a holistic run of the three scripts and the second script was run with two different parameter values (i.e., 2.5 and 5) twice. From Figure 4, we find that the second script that ran twice produced two versions of files at the same file location.

[2] https://github.com/yesworkflow-org/yw-idcc-17/tree/master/OHIBC_Howe_Sound_project/hybrid_provenance_queries/queries
[3] https://github.com/yesworkflow-org/yw-idcc-17/tree/master/OHIBC_Howe_Sound_project/multi_runs_retrospective_provenance_queries/queries
[4] https://github.com/yesworkflow-org/yw-idcc-17/tree/master/examples/C3C4/results

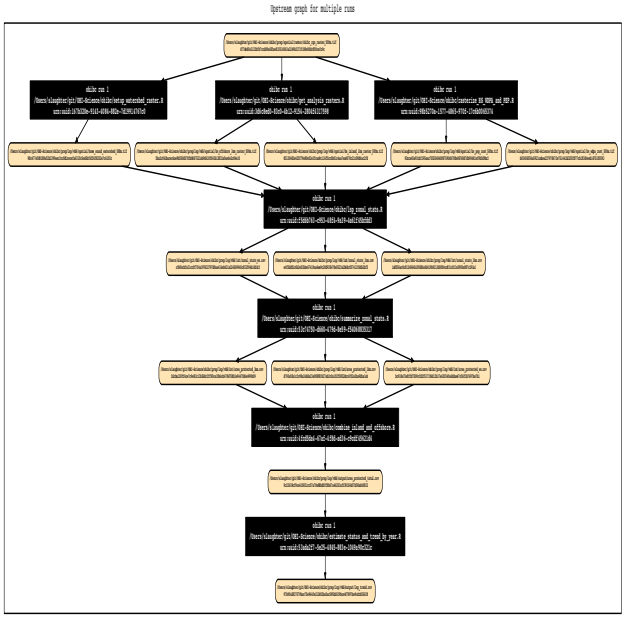



**Figure 3: Upstream subgraph that is relevant to "lsp_trend.csv" output for multi_scripts_multi_runs retrospective provenance for the OHIBC Howe Sound example**

**Reproducibility**. DataONE allows to make experiments reproducible in three ways. First, DataONE RunManagers capture both prospective and retrospective provenance and store related related provenance and metadata to database. RunManager database enables users to retrieval past version of used and generated data to support reproducibility. Second, DataONE data repository allows users to package workflow results in a structured DataONE data package and publish to a member node inside a DataONE federation network so that the workflow results (including scripts, data, provenance relationship, science metadata, etc.) can be displayed on DataONE data repository and be shared or linked with other scientific experiments in a larger scale. DataONE package contains provenance information (`prov:used`, `prov:wasGeneratedBy`, or `prov:wasDerivedFrom`) to describe the relationships between digital objects and data packages for reproducibility, instead of only encapsulating digital objects. Third, the query ability provided by RunManager can make users to reproduce and verify their scientific experiments as graphs without running the scripts.

## 4 CONCLUSIONS

Provenance is a useful component of the software development cycle, either to benefit researcher themselves or documenting and sharing code for use by future investigators. For the researcher, the quality of data produced by script runs can be assessed by identifying source data and verifying their quality. With our DataONE RunManager demo, we showcase how RunManager can capture, store, query, visualize, and publish prospective, retrospective and hybrid provenance. Also, we demonstrate how YesWorkflow can model a scientific workflow. Moreover, we demonstrated that provenance queries across heterogeneous provenance sources can yield a more complete and comprehensible picture of data provenance. We also think provenance describing datasets can facilitate data discovery for others.

## REFERENCES


[1] Roger S. Barga and Luciano A. Digiampietri. 2008. Automatic Capture and Efficient Storage of eScience Experiment Provenance. *Concurrency and Computation: Practice and Experience* 20, 5 (2008), 419–429. DOI: http://dx.doi.org/10.1002/cpe.1235

[2] Yang Cao, Christopher Jones, Vctor Cuevas-Vicenttn, Matthew B. Jones, Bertram Ludscher, Timothy M. McPhillips, Paolo Missier, Christopher R. Schwalm, Peter Slaughter, David Vieglais, Lauren Walker, and Yaxing Wei. 2016. DataONE: A Data Federation with Provenance Support.. In *IPAW (Lecture Notes in Computer Science)*, Marta Mattoso and Boris Glavic (Eds.), Vol. 9672. Springer, 230–234.

[3] Fernando Chirigati, Dennis Shasha, and Juliana Freire. 2013. ReproZip: Using Provenance to Support Computational Reproducibility. In *Proceedings of the 5th USENIX Conference on Theory and Practice of Provenance (TaPP'13)*. 1–1.

[4] Benjamin S. Halpern, Catherine Longo, Darren Hardy, Karen L. McLeod, Jameal F. Samhouri, Steven K. Katona, Kristin Kleisner, Sarah E. Lester, Jennifer Oleary, Marla Ranelletti, Andrew A. Rosenberg, Courtney Scarborough, Elizabeth R. Selig, Benjamin D. Best, Daniel R. Brumbaugh, F. Stuart Chapin, Larry B. Crowder, Kendra L. Daly, Scott C. Doney, Cristiane Elfes, Michael J. Fogarty, Steven D. Gaines, Kelsey I. Jacobsen, Leah Bunce Karrer, Heather M. Leslie, Elizabeth Neeley, Daniel Pauly, Stephen Polasky, Bud Ris, Kevin St Martin, Gregory S. Stone, U. Rashid Sumaila, and Dirk Zeller. 2012. An index to assess the health and benefits of the global ocean. *Nature* 488, 7413 (8 2012), 615–620. DOI: http://dx.doi.org/10.1038/nature11397

[5] D.N. Huntzinger, C.R. Schwalm, Y. Wei, R.B. Cook, A.M. Michalak, K. Schaefer, A.R. Jacobson, M.A. Arain, P. Ciais, J.B. Fisher, D.J. Hayes, M. Huang, S. Huang, A. Ito, A.K. Jain, H. Lei, C. Lu, F. Maignan, J. Mao, N. Parazoo, C. Peng, S. Peng, B. Poulter, D.M. Ricciuto, H. Tian, Xiaoying Shi, W. Wang, N. Zeng, F. Zhao, and Q. Zhu. 2014. NACP MsTMIP: Global 0.5-deg Terrestrial Biosphere Model Outputs (version 1) in Standard Format. (2014).

[6] Barbara Lerner and Emery Boose. 2014. RDataTracker: Collecting Provenance in an Interactive Scripting Environment. In *6th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2014)*. USENIX Association, Cologne.

[7] Timothy McPhillips, Tianhong Song, Tyler Kolisnik, Steve Aulenbach, Khalid Belhajjame, Kyle Bocinsky, Yang Cao, Fernando Chirigati, Saumen Dey, Juliana Freire, Deborah Huntzinger, Christopher Jones, David Koop, Paolo Missier, Mark Schildhauer, Christopher Schwalm, Yaxing Wei, James Cheney, Mark Bieda, and Bertram Ludäscher. 2015. YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts. *International Journal of Digital Curation* 10 (2015), 298–313. Issue 1.

[8] Leonardo Murta, Vanessa Braganholo, Fernando Chirigati, David Koop, and Juliana Freire. 2015. *No Workflow: Capturing and analyzing provenance of scripts*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 8628. Springer Verlag, 71–83. DOI: http://dx.doi.org/10.1007/978-3-319-16462-5_6

[9] Y. Wei, ShiShi Liu, D.N. Huntzinger, A.M. Michalak, N. Viovy, W.M. Post, C.R. Schwalm, K. Schaefer, A.R. Jacobson, C. Lu, H. Tian, D.M. Ricciuto, R.B. Cook, J. Mao, and X. Shi. 2014. NACP MsTMIP: Global and North American Driver Data for Multi-Model Intercomparison. (2014).


## A EXAMPLE RETROSPECTIVE PROVENANCE SUBGRAPH

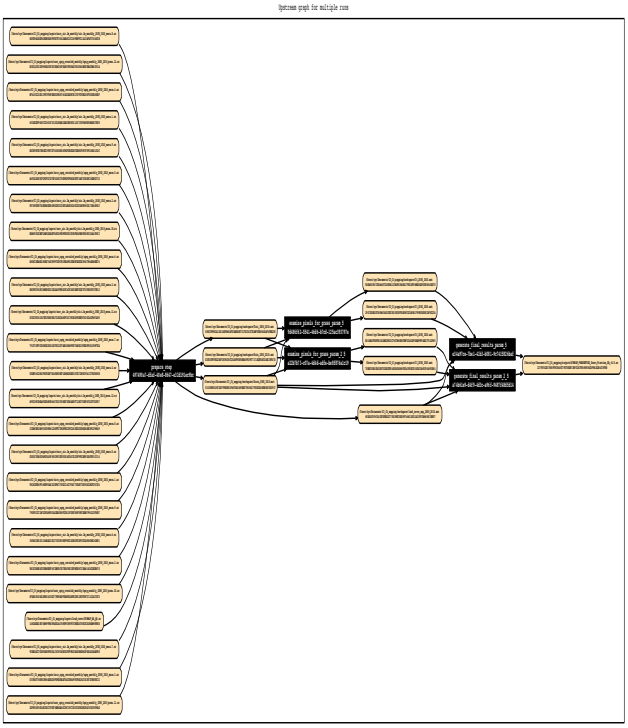



**Figure 4: Upstream retrospective provenance subgraph (longitudinal view) that is relevant to one output "`SYNMAP_PRESENTVEG_Grass_Fraction_NA_v2.0.nc`" for C3C4 version 2 example. In the diagram, an orange yellow box represents a data file and a black box represents an execution (a run).**